WinFS Data Model and Programming Model

Anil Nori
Architect

02/17/04

WinFS

WinFS ECS Talks

Overview	Quentin Clark
Data Model and Programming Model	**Anil Nori**
API	Mike Deem
Architecture and Implementation	Nigel Ellis
Schema	J. Patrick Thompson
Rules Engine	Praveen Seshadri
File System Integration	Sanjay Anand
Sync	Lev Novik

Data Requirements in Next Generation Applications

- Model complex objects
  - Complex structure
  - Inheritance
  - Unstructured, XML and Structured data
- Rich Relationships
  - Value-based
  - Link based
  - WinFS provides a built in model with more services for complex objects
- Rich and Common Query
  - Common across client and server
  - Common across different typed of data – SQL, Objects, XML
- Granular operations
  - Copy, Move
  - Backup/Restore
  - Security
- Rich organization
  - Hierarchical Namespace
- Active Notifications and Data Synchronization
- Integrated Business logic
  - Optimistic concurrency control and API mapping

WinFS Vision . . .

. . . is the active storage platform for organizing, searching and sharing all data

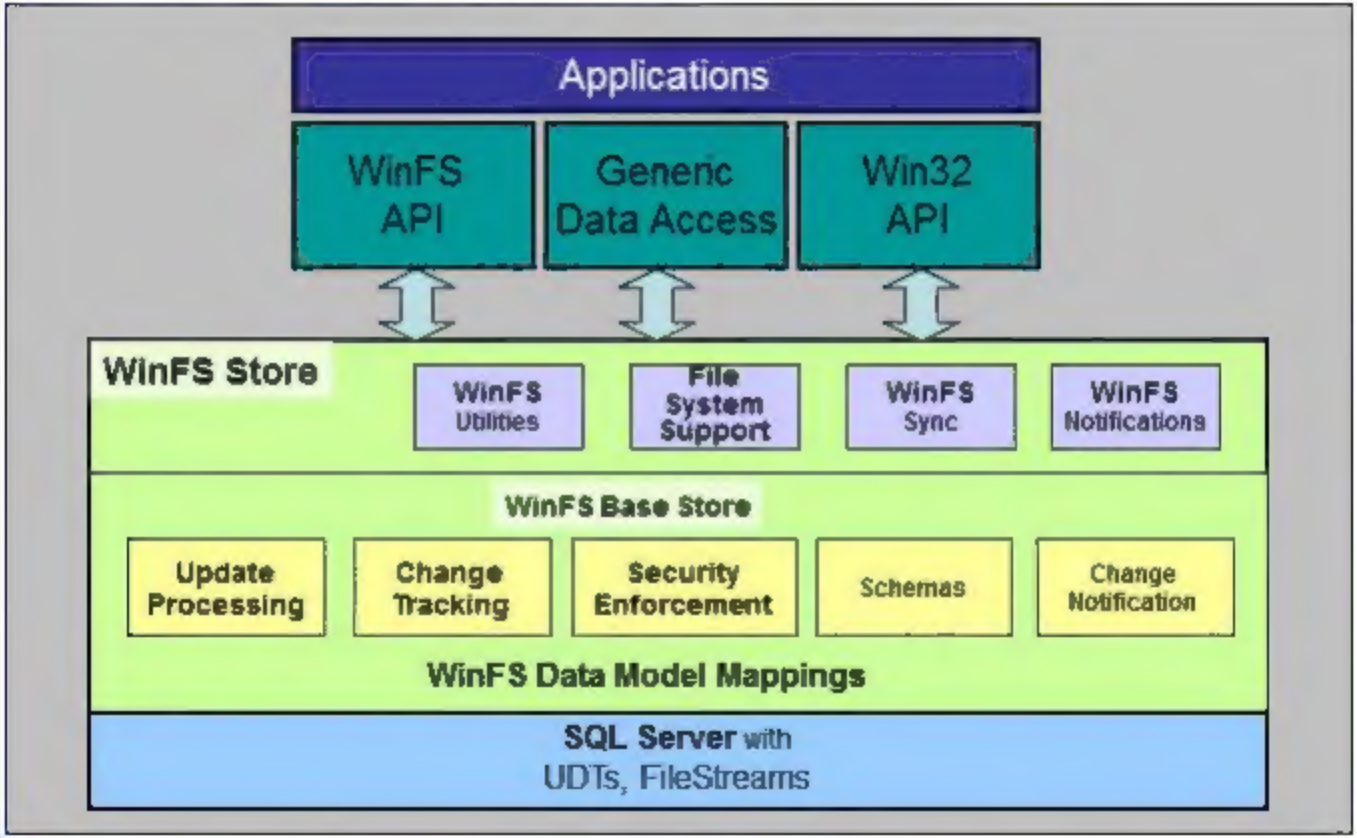
WinFS Enables ...

- Organization
  - Rich and varied organization beyond what a database system or a file system offers
  - Schemas, Tables, Rows, Items, Foldering, Namespaces, etc.
- Searching
  - Based on common schemas, structure of data, relationships, and organization
  - Over structured, unstructured, and XML data
- Sharing
  - Bridging the islands of data, WinFS ensures the right data is in the right place for the applications and users of that data
  - Through common storage, schemas, behaviors, synchronization and other services

WinFS *is* . . .

- About *Data*
  - Extending and broadening the data platform beyond file and Relational systems, and is the store for all types of data (structured, semi-structured, unstructured)
- *All* the data
  - WinFS makes data available to all appropriate locations, devices at all times
- *The* store
  - Authoritative store for the data
- A *platform*
  - Storage platform for application development
  - Rich schemas, programmability, extensibility
- *Active*
  - provides capabilities so that data can connect, flow and be more useful to users
  - Integrated rules, notifications, and workflow

WinFS Architecture



Data Model Agenda

- Overview
- Type System
- Items
- Relationships
- Extensions
- Views
- Indexing
- Data Model Mapping
- Open Issues
- Links and Aliases

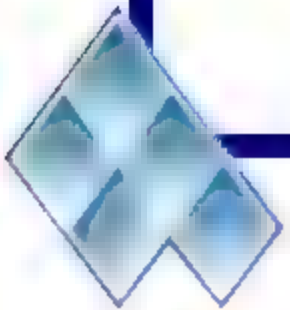
Data Model Goals

1. Enable modeling of schemas to represent everyday information:
   - People, Places, Events, Documents, Media, System data, etc.
   - Provide mechanisms to implement behavior
2. Provide organizational facilities for items
   - Relationships as first class entities to relate items
   - Queries as way to retrieve and classify items
   - Lists as a mechanism to explicitly organize items
3. Provide base primitives across all item types
   - Create, Update, Delete, Copy, Move, Serialize/De-serialize, Synchronization, Security, Notifications
4. Support schema extensibility

WinFS Data Model

- The WinFS Data Model describes
  - the shape of the data stored in WinFS
  - the constraints on the data
  - associations between data
- WinFS world is comprised of items, relationships and extensions
- Items are the primary objects that applications work on
- Items can be associated with other items via relationships
- Items can be extended with extensions
- WinFS data model provides the means to describe items, relationships and extensions and then operate against them

The WinFS Type System



WinFS

Type System

- The shape of the data stored in WinFS is defined using a type system
- All data stored in WinFS is an instance of a type in the WinFS type system
- The WinFS type system maps closely to the CLR type system

Schema

- WinFS Schemas serve as a way of grouping a set of WinFS type declarations within a schema namespace
- Defined using the WinFS Schema Definition Language
- A Schema contains definitions of:
  - Schema dependencies
  - WinFS Types
  - Views
  - Indexes
- Schemas are a unit of compilation, installation and versioning
  - Individual types in a schema can not be separately compiled nor installed
- System.Storage is the WinFS system schema namespace
  - Declares all WinFS scalar types
  - Declares the root types in the WinFS type system

Scalar Types

- WinFS uses a set of scalar types that closely match the CLR primitive types and maps to Managed SQL types:
  - Integral types: Byte, Int16, Int32, Int64
  - Real number types: Single, Double, Decimal
  - Boolean
  - DateTime
  - Guid
  - String and Binary – with size specification
  - Xml
  - Stream
- Declared in System.Storage namespace.
  - Only System.Storage namespace can declare scalar types

Enumerations

- Enumerations are scalar types with a set of named constants
- A WinFS enumeration type maps to the CLR enum type

```
<Enumeration Name="Gender" >
  <Value Name="Male" />
  <Value Name="Female" />
</Enumeration>
```

- The values of the enumeration are zero based
- A property of enumeration type is represented as Int32 value in the WinFS Store

Complex Types

- Complex type is defined with a name, a base type and a set of properties
- A property is defined with a name and a type, and can have additional attributes

```
<ItemType Name='Person' BaseType="System Storage.Item'>
  <Property Name= Name" Type="WinFS String"
            Size='256" Nullable="false" />
  <Property Name='Age' Type="WinFS Int32"
            Nullable='false" Default="1"/>
  <Property Name="Picture" Type="WinFS Binary"
            Size="max"/>
  <Property Name="Addresses" Type="MultiSet"
            MultiSetOfType="Address"/>
</ItemType>
```

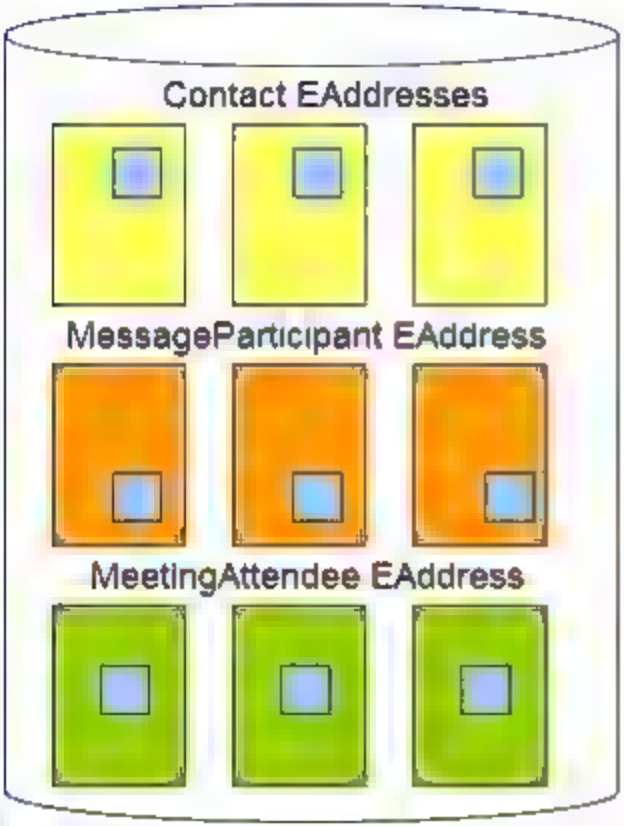
Property attributes

- Size attribute
  - Required if property is of type String and Binary
  - Value can be an integer or the literal "max" which indicates no size limit
- Nullable attribute
  - Indicates if the value of the property can be null
  - Valid only for properties of scalar types
- Default
  - Provides the default value for the property
  - Valid only for properties of scalar types
- Other attributes – ContentIndexed, ChangeUnit

Nested Types

- The types that are used as property types are called Nested Types
- System.Storage.NestedType is the root of the nested type hierarchy
- An instance of a nested type can only be stored as a value of a property of an instance of a complex type
  - The instance of the nested type is "nested" inside the complex type instance
- Nested type instances in WinFS are not globally searchable
  - An application can not retrieve all instances of a given nested type in the store

Nested Types Queries



- SELECT * FROM EAddress
  - Invalid query
- Valid queries

SELECT EAddresses
FROM Contact

SELECT EAddress FROM
MessageParticipant

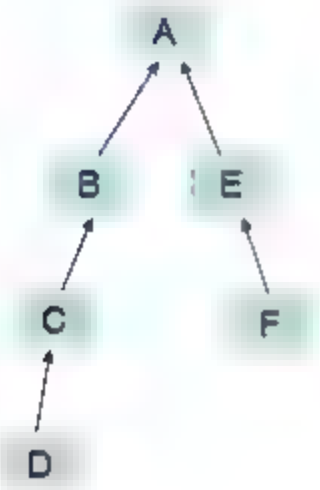
SELECT EAddress FROM
MeetingAttendee

Collections

```
<Property Name="Addresses" Type="MultiSet"
          MultiSetOfType="Address"/>
```

- MulitSet is a collection of complex nested type instances
- A MultiSet of (simple) scalar types is not supported
- The collection is ordered in the order of insertion
  - Members of the collection can not be reordered – the whole collection must be recreated
- Property of type MultiSet can not have null value
  - The collection always exists, but can be empty
- WinFS does not support constraints on the number of the members in a collection

Inheritance



- WinFS supports single-inheritance of complex types
- Inheritance terminology:
  - Type A is the **root** of the **type hierarchy**
  - Type B is a **derived type** from A
  - Type A is a **base type** of B
  - Types C, B and A are **ancestors** of type D
  - Types B, C, D, E, F are **descendants** of type A
  - An instance of type D is also an instance of types C, B and A. Type D is the **most-derived type** of the instance
- A derived type can not declare a property with a same name as a property of its base type

Items

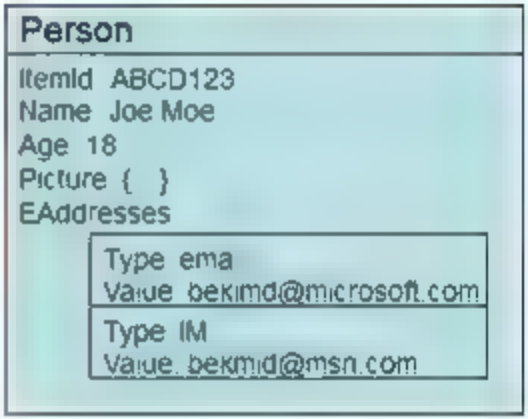
WinFS

Items

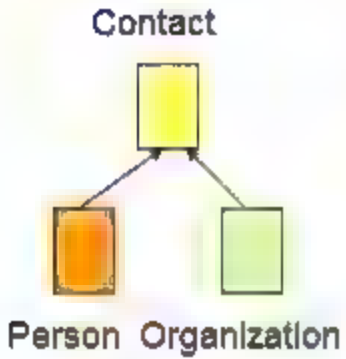
- Items are the primary objects that WinFS applications manipulate
- An item is an instance of an WinFS item type
  - An item type is a type that is a descendant of System.Storage.Item
- An item is uniquely identified by its ItemId
  - Serves as a primary key for items – it is guaranteed to be unique within a given WinFS store
  - It is non-nullable and immutable
- Applications can retrieve all instances of a given item type in a WinFS store
  - The query engine will return all instance of the given type and all its descendant types

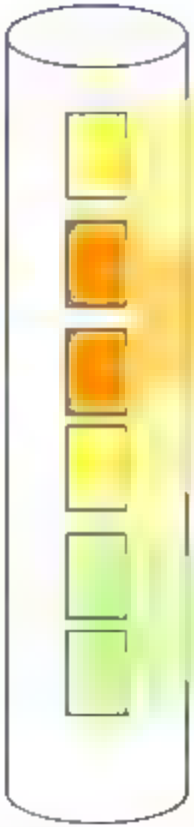
Item type declaration

```
<ItemType Name="Item" >
  <Property Name="ItemId"
    Type="WinFS.Guid" Nullable="false"/>
</ItemType>
<NestedType Name="EAddress
    BaseType="WinFS.NestedType" >
  <Property Name="Type"
    Type="WinFS.String" Nullable="false"/>
 <Property Name="Value"
    Type="WinFS.String" Nullable="false" />
</NestedType>
<ItemType Name="Person" BaseType="Item" >
  <Property Name="Name"
    Type="WinFS.String"
    Size="256" Nullable="false" />
  <Property Name="Age" Type="WinFS.Int32"
    Nullable="false" Default="1" />
  <Property Name="Picture"
    Type="WinFS.Binary"
    Size="max"/>
  <Property Name="EAddresses"
    Type="MultiSet"
    MultiSetOfType="EAddress" />
</ItemType>
```



Item Type Queries





SELECT * FROM Contact

SELECT * FROM Person

SELECT * FROM Organization

Items and Files

- Applications can use Win32 APIs to create files in WinFS
- If the file content is not promoted the file is represented as an item of type System.Storage.GenericFile
- If the file content is promoted the file will be represented as an item of a corresponding WinFS type (ex: Image, Document, AudioTrack, etc.)
- Win32 file and list attributes are not surfaced in the WinFS data model
  - Stored in separate tables
  - Accessible through Win32 APIs only

Relationships

WinFS

Relationships

- Relate two items
- Relationships enable
  - Item organization
  - Sharing across users and applications
  - Semantic relationships between items
- Organization
  - Shared:
    - Files In a folder
      - Folder is known as List
    - Songs In a Playlist
  - Exclusive:
    - Compound item
      - Email and its attachments
- Semantics Relationship
  - DocumentAuthor
  - Dynamically specified

Organization: Sharing

- Holding Relationship
  - Constructs the namespace
  - Propagate security
  - Multiple holding relationships from containing lists to items possible
  - Only within a single store
  - Both related items can be independently modified
  - Only from list to items
- Support WinFS (Win32) namespace and security
- Items are organized using holding relationships
- Legacy files (in a folders) are modeled using holding relationships

Organization: Exclusive

- Embedding Relationship
  - Integrity unit: item and its embedded items
  - Embedded item cannot be independently modified
  - Lifetime is controlled by the parent item
  - Embedding relationships form a tree
  - If root of embedding tree deleted, all items are atomically deleted
  - Within a single store
  - A subitem can be removed from embedded item tree
  - Compound item identified by ItemId of parent item
  - e.g. Compound Documents and Email & attachments

Semantic (Dynamic) Relationships

- Reference Relationship
  - Explicit Relationship without lifetime control
  - To any item (within and across stores)
  - e.g. "Author" Relationship: Document and Contact

Relationships

- WinFS supports binary typed relationships between two items – source and target item
- The source and target items are identified by their ItemId's
- Source item manages the lifetime of the relationship instance
  - The relationship instance can not exist without the source item
- Relationship types are descendents of System.Storage.Relationship type
- A Relationship type is declared as a complex type, plus:
  - Names of the source and target endpoint
  - Constraints on the type of the source and target item
  - Allowed relationship instance modes
    - Holding, Embedding and Reference

System.Storage.Relationship Type

- Properties of System.Storage.Relationship type:
  - SourceItemId – the ItemId of the source item,
  - RelationshipId – the Id of the relationship,
  - TargetItemId – the ItemId of the target item,
  - Mode – indicates the mode of the relationship:
    - Holding, Embedding or Reference
  - Name – name of the relationship,
    - Can not be null for holding relationships, must be null for non-holding relationships
    - The name is unique for all holding relationships for a given source item
  - IsHidden – can be set to indicate to applications that the relationship should not be displayed to the end user
- (SourceItemId, RelationshipId) is a primary key for relationships
- SourceItemId, RelationshipId, TargetItemId and Mode are immutable

Relationship Modes

Mode	Organization	Endpoint Type constraints	Target locality	WinFS Namespace	Graph type
Ho d ng	Shared	Enforced on source and target	Local store	Forms the WinFS namespace	DAG
Embedd ng	Exc us ve	Enforced on source and target	Local store	No	Tree
		only on source	store		

Item Lifetime

- Holding relationships enable sharing. However, they should not prevent appropriately entitled user to delete items
- Also, in a shared environment, independent removal of holding relationships should not accidentally delete items
- Removing all holding relationships to an item does not automatically delete the item
- An item is always reachable from the store root
- An item can only be deleted explicitly by performing a delete operation on the item
- Explicitly deleting an item yields holding relationships to the item dangling (target set to null)

Item Domains, Regions

- Starting at an item, the item domain is the set of items reachable by traversing the holding links
- Item domains are useful to restrict the search (name) space
- An item domain provides the scope for queries. The items in the query result must be in the item domain
- There is always a path using holding relationships from the store root to every item in the store

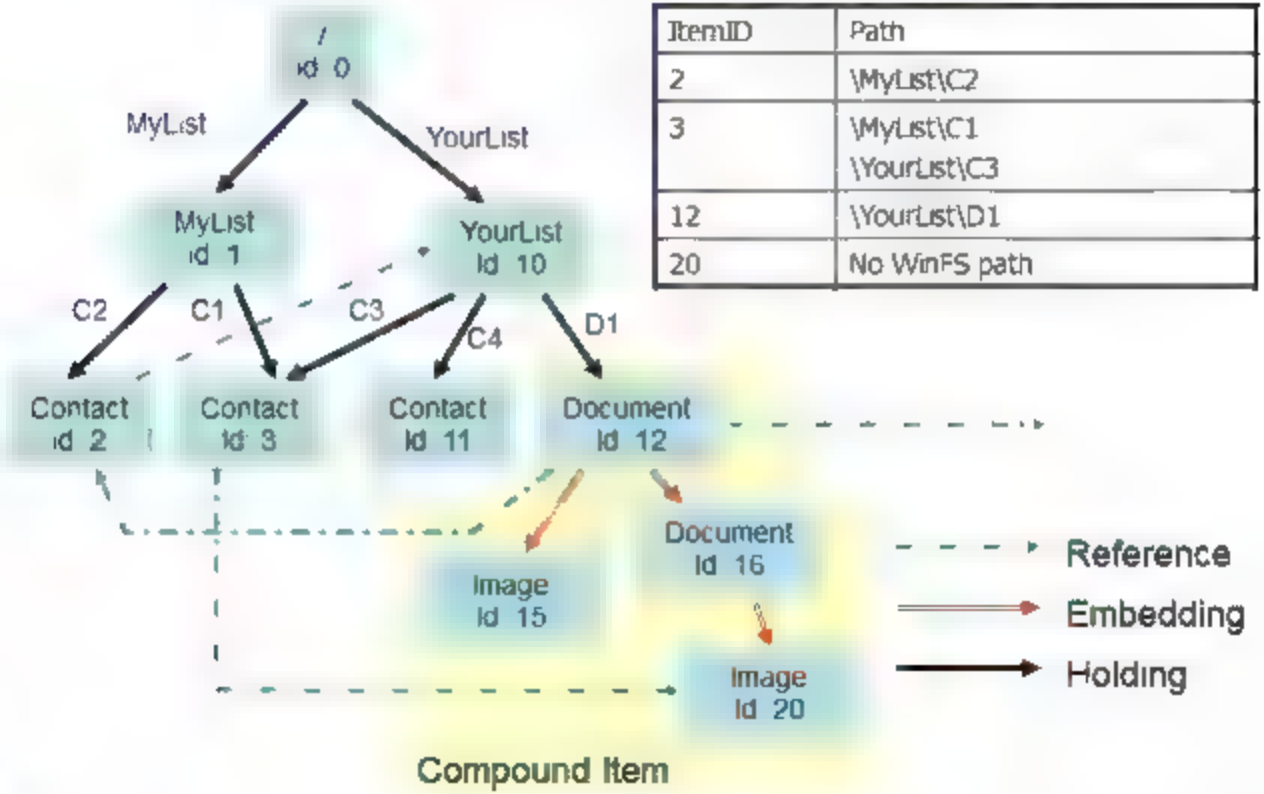
Regions

- It is useful to partition (not a strict partition) the store into regions. However, these regions can be overlapping. An item can live in multiple regions
- A Region is a special item domain rooted in an item that is marked as a region root. The store root is always a region root. There can be other region roots, e.g. User Roots
- Regions ensure that accidental removal of holding relationships do not delete the items
- Items that are not in any lists will be reachable from region roots. There will always be a path from the region root to items in that region, e.g. all items in AnilNori Region.
- When the last holding relationships (from a list) to an item is removed, implicitly a holding relationship is created from the region root to the item. This will ensure that items in a region always reachable from the region root
- It is possible for an item to get out of a region. But the item is always in the store root region

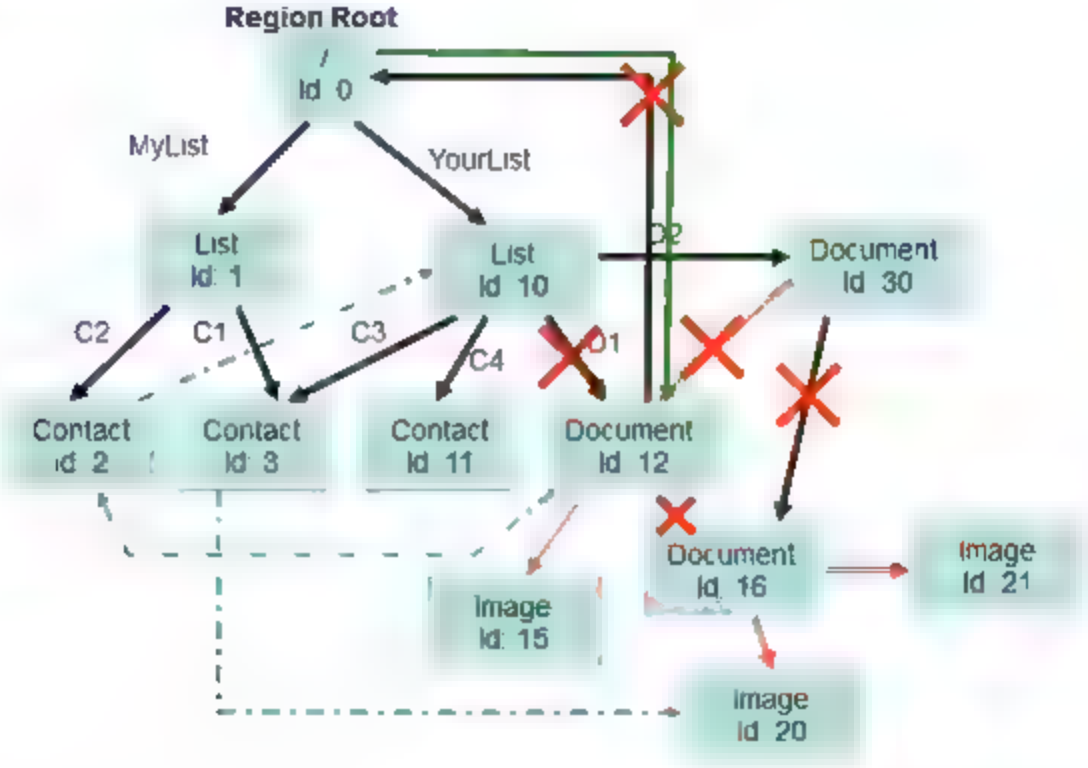
Relationship Modes Examples

WinFS namespace paths

ItemID	Path
2	\MyList\C2
3	\MyList\C1 \YourList\C3
12	\YourList\D1
20	No WinFS path



Relationship Modes Example



Relationship type inheritance

- Derived relationship type can further restrict the types of the endpoints
  - The declarad endpoint type must be a descendant of the endpoint type declared in the base relationship type
- Derived relationship type can further restrict the allowed modes
  - The derived relationship can only dissallow modes that the base type allows
  - The derived relationship can not allow modes that the base relationship dissallows
- Applications can efficiently retrieve all instances of a given relationship type in a WinFS store
  - The query engine will return all instance of the given type and all its descendant types

Relationship Type Declaration

Extensions

WinFS

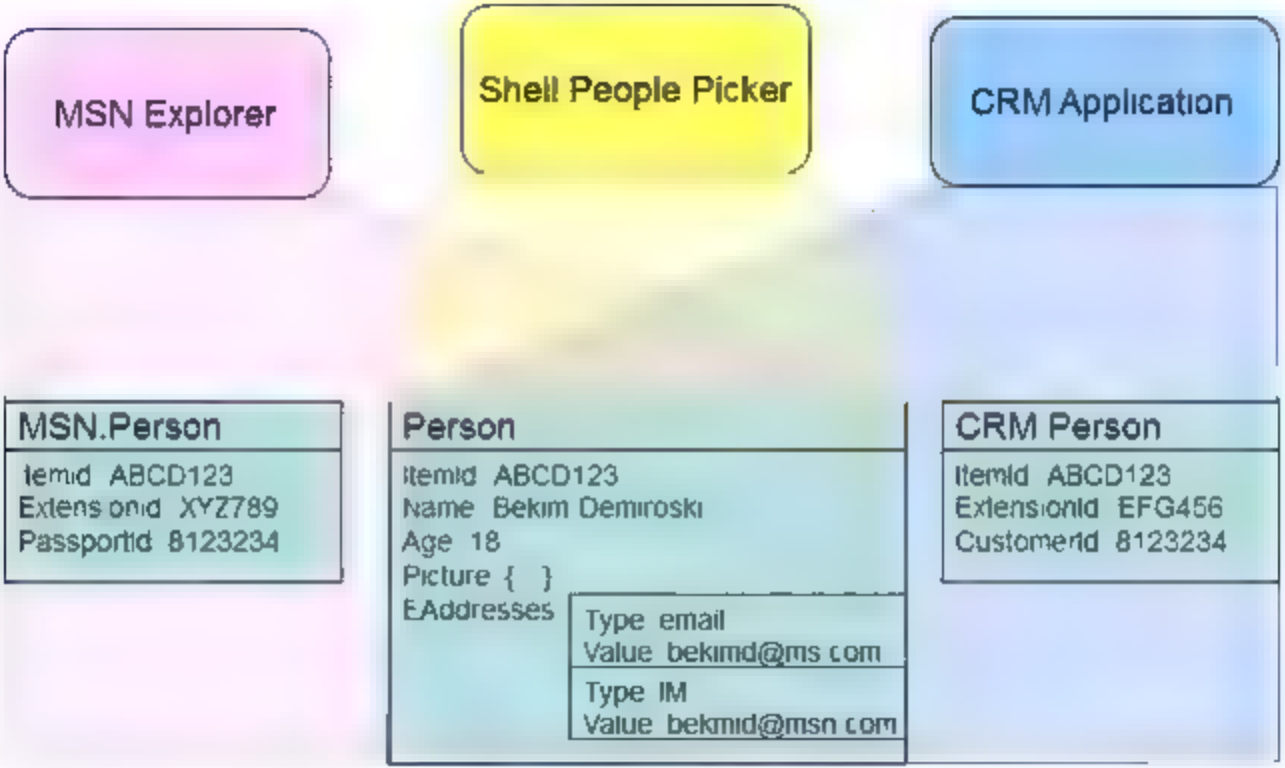
Item Extensions

- Extensions enable adding of additional properties to existing items
  - Extensions are a primary mechanism for ISV to extend instances of existing item types
  - Extensions are also a form of multi-typing
- An extension is an instance of an extension type – a descendant of System.Storage.Extensions
- System.Storage.Extension defines two immutable properties
  - ItemId is the id of the item that the extension is associated
  - ExtensionId is a unique identifier of the extension relative to ItemId
- (ItemId, ExtensionId) is the primary key of extension

Item Extensions

- An extension instance is always associated with an item
  - An extension instance can not exist if an item with the same ItemId doesn't exist
  - The item controls the lifetime of the extension
- Only one instance of any of the types in an extension type family is allowed on a given item
  - Extension type family is rooted in a type that derives from System.Storage.Extension
  - The extension can contain a MultiSet property if collection semantics is required
- Instance of an extension type can be associated with an instance of any item type
  - E.g. Sticky Notes
- Extension can not be source or target of relationships

Item extensions



WinFS Services

- Security
- Serialization
- Copy
- Backup/Restore
- Synchronization
- Notifications

Views

WinFS

Views

- WinFS supports view declaration using the power of SQL views
- A WinFS View declares
  - A set of columns that map to the columns of the select statement
  - A SELECT query which can reference other views or WinFS item, extension or relationship views
- Column data type can be any WinFS scalar, item, nested, extension or relationship type
- Views are SELECT-only
  - No DML is allowed
  - No triggers are supported

View Example

```
<View Name="Document">
  <Column Name="Document" Type="Core.Document" />
  <Column Name="Author" Type="Core.DocumentAuthor" />
  <Column Name="Contact" Type="Core.Contact" />
  <Query>
    SELECT Core [Document] _Item,
     Core [Relationship'DocumentAuthor] _Relationship,
     Core [Contact]._Item
    FROM Core [Document],
     Core [Relationship'DocumentAuthor], Core [Contact]
    WHERE Core [Document] ItemID =
     Core [Relationship'DocumentAuthor] Source AND
    Core [Relationship'DocumentAuthor] Target ItemID =
     Core.[Contact] ItemID
  </Query>
</View>
```

Indexing

WinFS

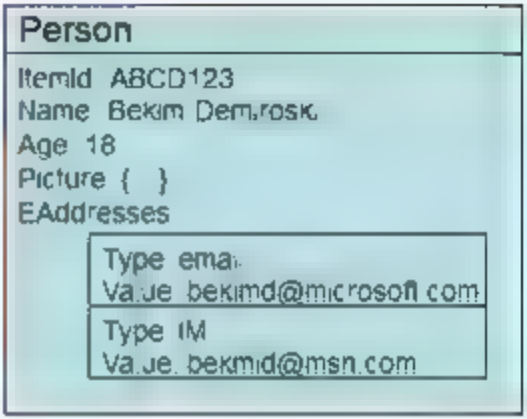
Indexing

- WinFS supports indexing on properties of Items, Relationships and Extensions.
  - Allows for efficient search and retrieval
  - Indexed properties could be nested to any level of depth
  - Indexes on properties of a collection property are allowed
  - Not supported for a collection within a collection

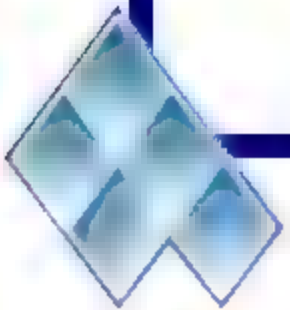
Index declaration example

```
<Index Name="PersonByName"
    Type="Person">
    <IndexField Property="Name"/>
</Index>

<Index Name="PersonByEaddress"
    Type="Person">
    <IndexCollection
        Name ="EAddresses">
        <IndexField Property="Type">
        <IndexField Property="Value">
    </IndexCollection>
</Index>
```



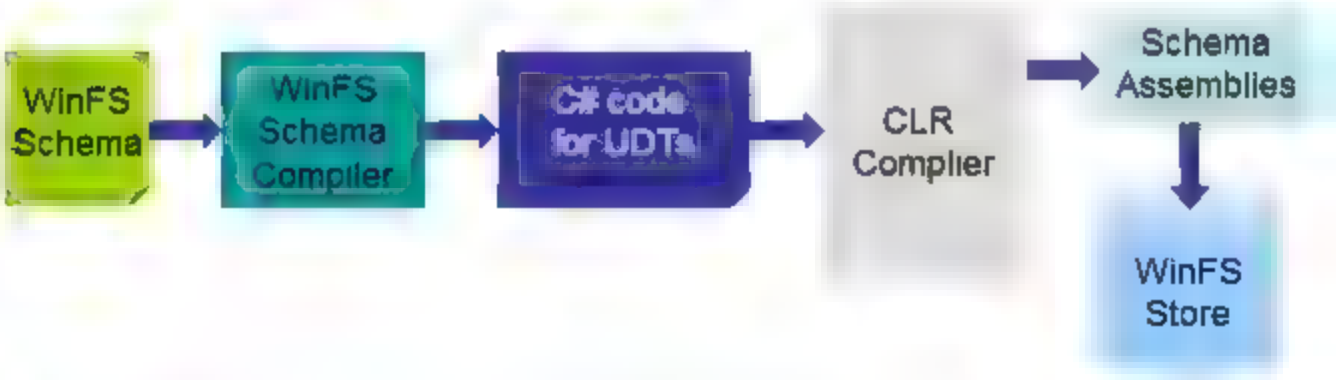
Implementing the WinFS Data Model in the WinFS Store (overview)



WinFS

Schema compilation process

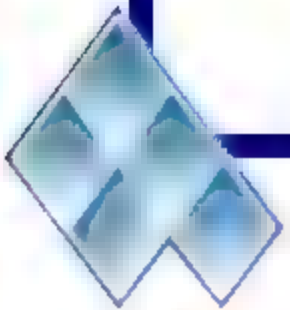
- WinFS schemas are defined using XML syntax
- The WinFS schema compilation process generates C# code from the WinFS Schema file
- The C# source files are compiled into assemblies
- The assemblies are installed into a WinFS store
  - WinFS types are registered as UDTs
  - Views and other database objects are created



Data Model Mapping Overview

- A WinFS schema is mapped to a SQL schema
- A CLR class is generated for each Item, Nested, Extension Relationship type
- The classes are registered as SQL User Defined Types (UDTs)
- Search views are provided for each Item, Extension and Relationship type
- Updates are enabled through the WinFS Update API operations
  - CreateItem, CreateRelationship, CreateExtension
  - UpdateItem, UpdateRelationship, UpdateExtension
  - DeleteRelationship, DeleteExtension

Programming Model



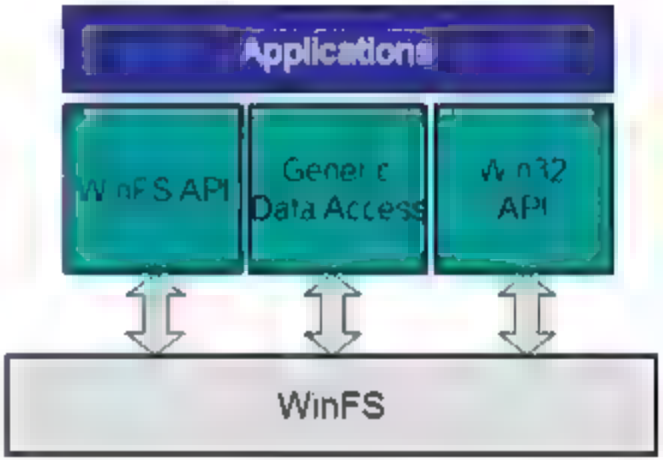
WinFS

WinFS Programming Model

- WinFS Programming Surface
- WinFS APIs
- Query language
- Supporting XML, File data
- Business Logic

WinFS Programming Surface

- WinFS provides multiple programming surfaces:
  - Object APIs for accessing items
  - SQL Access (read) to items
  - Win32 access for file access
  - APIs for accessing XML and File data
- Item (WinFS) API provides strongly typed data classes for items
- Generic read data access available through ADO.Net
- Win32 API offers file stream and directory access



WinFS Item APIs

- Support the WinFS data model concepts
- Provide great "out of the box" experience
- Evangelized API for ISV's
- SQL query and view abilities for ADO.NET direct (but no support for updates)
- WinFS API has matured into a full service framework
- WinFS services (like Sync and Notifications) are exposed
- Application logic is factored into the WinFS API classes

Key Concepts of WinFS API

- Query
  - Query expressions on Objects, OPATH
  - Flexible, expressive, powerful
- Navigation
  - WinFS data is a graph of items
  - Use relationships between items to traverse the graph
- Actions
  - Common actions: Create, Delete, Update
  - Domain specific: SendMail, CheckFreeBusy
  - A client side business logic framework
- Events
  - Subscribe to changes
  - Handling notifications
  - Watchers
- Rich Application Views

Query Language

- WinFS supports OPath (a filter-based) query language in the API and SQL in the Store
- OPath is mapped to SQL and executed in the store
- OPath is expressed over the API classes
- OPath simple and expression oriented
  - Not as rich as SQL
  - Eventually enhance with more SQL capabilities
- OPath integrates well with C#

XML Support: Mechanisms

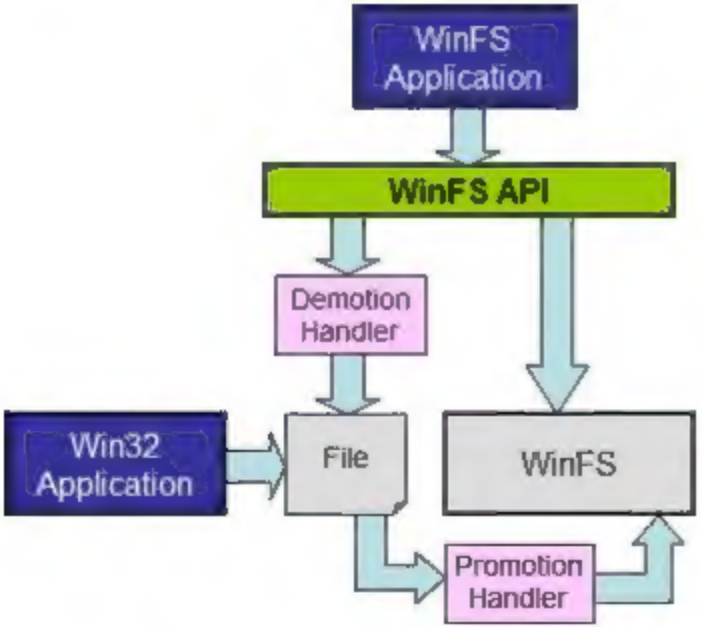
- WinFS is not an XML Store
- WinFS can store and query XML content
- XML datatype
  - Can define as part of WinFS Type, as a property
  - Indexed
  - XPath/XQuery on XML datatypes
- WinFS APIs to access XML content
- XML representations of WinFS objects
  - Read and Write
  - WinFS Type installation produces XSD
- Import/Export uses XML
  - Heavier than just XSD representations of Items to enable full-fidelity graph round-trips

XML Properties

- Store XML as a part of an Item, Relationship or Extension
- Property value does not have to conform to a schema
  - Application can store schemas and validate as needed
- May have multiple XML properties in a type
  - Flexible data modeling and document structures
- Use XML property values in query conditions
  - XQuery can be combined with predicates on other properties
  - XML is stored in a pre-parsed format for more efficient query
- Can project XML data into a query result
  - Projection allows for efficiently retrieving parts of objects
  - Querying for whole objects is a simple "automatic" projection
- Can access XML via XmlReader and XmlWriter

File Data

- WinFS is an item store, but stores files as well
- WinFS provides Win32 API support for file access



Business Logic

- Behaviors are functions tightly coupled to a data model.
  - Like OOP methods or messages
  - The set of behaviors defines an interface and makes a strong abstraction barrier
- A type designer sets up
  - The data model (schema) for an Item
  - Behavior binding points (i.e. the behavior interfaces)
  - Default implementations of the behaviors
- A behavior provider supplies
  - New implementations of the behaviors
  - Additional private data
- A consumer of an Item type
  - Knows the schema
  - Can call the public interfaces

Simple Example

Type Designer

```
<Schema Name="FooSchema">
 <NestedType Name="FooBehavior" BaseType="System.Storage.Behavior"
   Abstract="true"/>
 <ItemType Name="FooItem" BaseType="Item">
   <Property Name="Behavior" Type="FooBehavior" />
 </ItemType>
</Schema>

namespace FooSchema {

 public abstract class FooBehavior : System.Storage.Behavior {
   public abstract void DoFoo();
 }

 public class FooItem : Item {
   public FooBehavior Behavior { get; set; }

 public void DoFoo() {
  if( Behavior != null ) Behavior.DoFoo();
 }

}
```

```
// Usage example: added behavior to an item
FooItem f = new FooItem();
f.Behavior = new MyBehavior();
ic.SaveChanges();
```

```
// Usage example: executing behavior
FooItem f = ic.FindItemById( fid ) as FooItem;
f.DoFoo();

Output:
Doing Foo
```

Behavior Designer

```
<Schema Name="MySchema">
 <NestedType Name="MyBehavior" BaseType="FooSchema.FooBehavior" />
</Schema>

namespace MySchema {
 public class MyBehavior : FooSchema.FooBehavior {
  public override void DoFoo() { Console.WriteLine( "Doing Foo" ); }
```

Italic = generated code

Bold = Hand Written Code

Q/A ?